GPU Projection of ECAS-II Segmenter for Hyperspectral Images Based on Cellular Automata

Javier López-Fandiño, Blanca Priego, Dora B. Heras, Francisco Argüello, and Richard J. Duro

***Abstract*—Segmentation is a key issue in the processing of multidimensional images such as those in the field of remote sensing. Most of the segmentation algorithms developed for multidimensional images begin by reducing the dimensionality of the images, thus loosing information that could be relevant in the segmentation process. Evolutionary cellular automata segmentation (ECAS-II) is an evolutionary approach that provides cellular automata-based segmenters considering all the spectral information contained in a hyperspectral image without applying any technique for dimensionality reduction. This paper presents an efficient graphics processor unit implementation of the type of segmenters produced by ECAS-II for land cover hyperspectral images. The method is evaluated over remote sensing hyperspectral images, introducing it on a spectral–spatial classification scheme based on extreme learning machines. Experiments have shown that the proposed approach achieves better accuracy results for land cover purposes than other spectral–spatial classification techniques based on segmentation.**

***Index Terms*—Cellular automata (CA), CUDA, evolutionary cellular automata segmentation (ECAS-II), extreme learning machines (ELM), graphics processor unit (GPU), hyperspectral images, segmentation.**

I. Hyperspectral Remote Sensing Image Processing Background

HYPERSPECTRAL imagery comprises the reflectance values over a wide wavelength range for each pixel [1], [2]. The huge amount of spectral information present in this kind of images provides useful capacities that can be exploited in all kinds of remote sensing applications including segmentation, classification, target detection, change detection, or anomaly detection [3], [4]. This work is focused on segmentation produced by the evolutionary cellular automata segmentation (ECAS-II) algorithm [5] and applied to land cover classification images. They allow the segmentation of the image, while preserving all the spectral bands from the original dataset.

This work was supported in part by the Ministry of Science and Innovation, Government of Spain, co-founded by the FEDER funds of European Union, under contract TIN2013-41129-P and TIN2015-63646-C5-1-R, and by Xunta de Galicia, Program for Consolidation of Competitive Research Groups 2014/008 and GRC 2013-050. The work of J. López-Fandiño was supported by the Xunta de Galicia under a predoctoral grant. *(Corresponding author: Javier López-Fandiño.)*

J. López-Fandiño, D. B. Heras, and F. Argüello are with the Centro Singular de Investigación en Tecnoloxías da Información, Universidade de Santiago de Compostela 15782 Santiago de Compostela, Spain (e-mail: javier.lopez.fandino@usc.es; dora.blanco@usc.es; francisco.arguello@usc.es).

B. Priego and R. J. Duro are with the Integrated Group for Engineer- ing Research, Universidade da Coruña, 15001 A Coruña, Spain (e-mail: blanca.priego@udc.es; richard.duro@udc.es).

Different dimensionality reduction techniques are usually applied during the segmentation process. Principal component analysis (PCA), a procedure to obtain a set of linearly uncorrelated variables from a larger set of possibly correlated variables through an orthogonal transformation, is computed in [6] to build the morphological profiles and is also used in [7]. Independent component analysis (ICA), which is a statistical method for transforming an observed multidimensional vector into components that are as statistically independent as possible, can also be used instead of PCA as an input to later processing stages. For instance, Dalla Mura, Villa, Benediktsson, Chanussot, and Bruzzone[8] use ICA as a preprocessing tool for a later spectral–spatial classification using extended morphological attribute profiles and support vector machines (SVM). A wavelet transform is used in [9] and [10] to reduce the high dimensionality of hyperspectral imagery. A robust color morphological gradient is used in [11] to reduce the hyperspectral dataset to a one-band image. Nevertheless, there are other methods that do not apply any dimensionality reduction, and their results have proven that it is desirable to preserve all the spectral information in the segmentation process. The hierarchical segmentation algorithm (HSeg) [12] uses hierarchical stepwise optimization to produce connected regions. Minimum spanning forest (MSF) is used in [13] and combined with a random marker (RD) in [14] in a spectral–spatial scheme including the use of SVM and majority vote.

ECAS-II [5] is an evolutionary scheme that produces segmenters based on cellular automata (CA). The objective of the resulting automaton is to perform a segmentation of the hyperspectral dataset through progressive modifications of the spectrum of each pixel depending on the spectra of the pixel's neighbors. Each segmenter produced by ECAS-II is especially adapted to performing the segmentation in a particular way given by an automatically generated set of automaton rules. These segmenters make use of gradients with respect to neighboring pixels and progressively modify the spectra of the pixels through successive iterations of the CA, until they fit the segmentation characteristics present in a training set. This training set can be of much lower dimensionality than the images to be processed. Thus, ECAS-II works with all of the spectral information without any type of projection onto lower dimensionalities.

SVM [15]–[17] are traditionally employed for the classification of hyperspectral datasets. Nevertheless, to achieve real-time processing, faster methods than SVM are needed. Extreme learning machines (ELM) are a good alternative previously used in remote sensing that improves SVM runtimes achieving competitive accuracy results compared to SVM [18]–[20]. In this work, we apply ELM [21] to the previously segmented image in order to achieve the final classification map.

The techniques to deal with classification and segmentation of hyperspectral datasets present high computational requirements [22]. Recently, some spectral–spatial schemes have been projected in graphics processor unit (GPU) to minimize their execution times. A spectral–spatial scheme in GPU comprising a watershed-based segmentation combined with SVM through a majority vote technique is introduced in [23]. A similar scheme that relies on ELM and also includes a final spatial regularization based on neighborhood pixels is presented in [21]. The strategy presented in [10] is based on wavelets, extended morphological profiles and SVM. The segmenters obtained by the ECAS-II algorithm presented in this paper are suitable for projection on GPU, since their CA operate independently over each pixel of the hyperspectral dataset, similarly to the CA presented in [24].

This paper involves two main goals. On the one hand, the presentation of the first GPU implementation of the segmenters produced by the ECAS-II algorithm. On the other hand, the combination of these segmenters with an ELM-based classification algorithm in order to improve the final classification accuracy results through a spectral–spatial classification scheme similar to [21]. The results will be compared in terms of accuracy to those obtained by other spectral–spatial classification schemes presented in the literature that also incorporate spatial information by segmentation. Execution times in GPU will also be compared to an optimized CPU implementation of the algorithm.

The remaining sections of this paper are organized as follows. Section II introduces the algorithm employed to segment hyperspectral datasets. Section III details the CUDA GPU implementation of the algorithm. The experimental results are discussed in Section IV. Finally, Section V summarizes the conclusions of this work.

II. ECAS-II-Based Spectral–Spatial Classification of Hyperspectral Images

This section introduces the ECAS-II segmentation algorithm [5], whose GPU implementation will be studied in Section III.

The ECAS-II segmentation algorithm is an iterative method that segments hyperspectral images preserving the original dimensionality of the image data. To achieve this, for each iteration of the algorithm, the spectrum of each cell (pixel) is combined with the spectra of some of its neighbors, through weighted averaging. The selection of the neighboring pixels that will contribute to the final combined spectrum depends on a set of transition rules controlling the operation of the CA and which determine the complexity and behavior of the segmentation algorithm. Once the CA is iteratively applied to the datacube, the final output will be a transformed hyperspectral image, in which pixels that belong to the same class share a similar spectrum. This means that the hyperspectral image is implicitly segmented but, unlike most segmentation methods, the final datacube preserves the complete wealth of spectral bands avoiding the loss of spectral information. In other words, the segmentation is carried out in terms of spectra and not of any kind of projection or lower dimensional representation.

The segmentation algorithm is described in Fig. 1. The algorithm comprises three main stages. As a first step, some

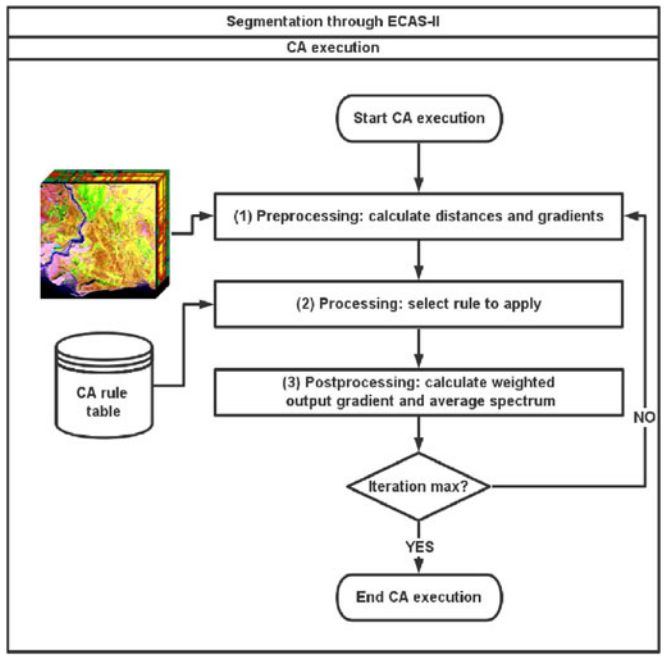

Fig. 1. ECAS-II segmentation algorithm flowchart.

neighborhood information is extracted for each cell (each cell maps to a pixel i whose state S_i is given by the B-band spectrum of the pixel). This information is represented in the form of spatiospectral gradients taking into account the pixels contained in three different $N \times N$ bidimensional windows, where $N \in \{3, 5, 7\}$. The computation of the spatiospectral gradients requires the calculation of the spectral distance between each pixel and all of its neighbors in a given window. For this purpose, the normalized spectral angle distance has been chosen

$$\alpha_{i,j} = \frac{2}{\pi} \cos^{-1} \left(\frac{\sum S_j S_i}{\sqrt{\sum S_j^2} \sqrt{\sum S_i^2}} \right) \in [0, 1] \qquad (1)$$

where i is the reference pixel, j is a pixel in the neighborhood window, $\alpha_{i,j}$ is the spectral angle for cell i with respect to cell j, and the summation is performed over the components of the state of S_i, i.e., the spectral dimension of a pixel.

The spectral angle has been widely used as a spectral similarity measure for material identification [25], [26]. The main advantage of using this spectral distance is that it is independent of dimensionality, which enables the method to be applied over Red, Green, Blue, multispectral, or hyperspectral images.

Then, for each pixel i, three spatial–spectral gradients (being $G_{X_N}(i)$ the horizontal component and $G_{Y_N}(i)$ the vertical component) are calculated using the spectral distances that were previously obtained and considering the three spatial window sizes $N \in \{3, 5, 7\}$:

$$G_{X_N}(i) = \sum_{j=1}^{N \cdot N} \alpha_{i,j} M_{X_{N_j}}, \quad G_{Y_N}(i) = \sum_{j=1}^{N \cdot N} \alpha_{i,j} M_{Y_{N_j}} \qquad (2)$$

where $M_{X_{N_j}}$ and $M_{Y_{N_j}}$ represent the jth elements of the horizontal/vertical gradient masks M_{X_N} and M_{Y_N}.

As three different window sizes are considered in this version of the ECAS segmentation algorithm, the gradient calculation will lead to three moduli ($|G_N(i)|$) and angles ($\phi_N(i)$), expressed as

$$|G_N(i)| = \sqrt{G^2_{X_N}(i) + G^2_{Y_N}(i)},$$
$$\phi_N(i) = \tan^{-1}\left(\frac{G_{Y_N}(i)}{G_{X_N}(i)}\right). \quad (3)$$

The second step of the method consists in selecting one rule from the set of transition rules that make up the CA based on the information of gradient moduli and gradient angles.

The set of transition rules comprises a group of M rules, each one containing six parameters:

$$CA = \begin{pmatrix} |\mathfrak{G}_{3,1}| & |\mathfrak{G}_{5,1}| & |\mathfrak{G}_{7,1}| & \Phi_{5,1} & \Phi_{7,1} & \theta_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ |\mathfrak{G}_{3,k}| & |\mathfrak{G}_{5,k}| & |\mathfrak{G}_{7,k}| & \Phi_{5,k} & \Phi_{7,k} & \theta_k \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ |\mathfrak{G}_{3,M}| & |\mathfrak{G}_{5,M}| & |\mathfrak{G}_{7,M}| & \Phi_{5,M} & \Phi_{7,M} & \theta_M \end{pmatrix} \quad (4)$$

where the first five parameters of each rule in (4) correspond to a direct representation of spatiospectral gradients, three gradient moduli ($\mathfrak{G}_{N,k}$), and two gradient angles ($\Phi_{N,k}$), $k \in [1, M]$. The angle of the gradient for the window of size 3 ($\Phi_{3,k}$) has not been included because it is considered equal to 0 with the aim of making the three vector gradients, as a group, independent from angle rotations and reflections. Then, the selection of one rule from the ruleset is performed by comparing the neighborhood information (moduli and angles of the spatiospectral gradients) to the first five parameters of each one of the M rules and selecting the closest in terms of Euclidean distance.

The last parameter of each rule (θ_k) contains the information needed to update the spectrum of the pixel. This parameter determines which of the neighboring pixels will contribute to the updated pixel's spectral average.

The transformed image is calculated by applying a filter to the spectra of the neighborhood pixels (stage 3 in Fig. 1) so that the larger the distance to the central pixel, the smaller the weight of the pixel. This kind of filter is used because classification accuracies can be improved when a smoothing is applied to the hyperspectral image [21], [27]. In this paper, different filters assigning weights that are inversely proportional to the distance have been tested with different distance measures to achieve the best accuracy results. The transformed image is then taken as the input to the next iteration of the process or as the final output if the maximum number of iterations was reached.

The segmented hyperspectral datacube obtained after the application of the ECAS-II segmenter could be used as input to other processing tasks, such as the labeling of the datacube obtaining a 2-D classification map.

In this work, we will combine the segmentation map obtained with an ELM classification process [21] seeking the spectral–spatial classification of remote sensing hyperspectral datasets.

III. ECAS-II-Based Spectral–Spatial Hyperspectral Image Classification in GPU

In this section, we introduce some CUDA programming fundamentals as well as the CUDA implementation of the algorithm proposed in Section II for the segmentation of hyperspectral datasets.

A. CUDA GPU Programming Fundamentals

Commodity GPUs provide massively parallel processing capabilities based on their data parallel architecture. CUDA is a combination hardware/software platform that enables NVIDIA GPUs to execute programs invoking parallel functions called kernels that execute across many parallel threads [28]. These threads are organized into blocks so that each thread executes an instance of the kernel following an SIMD programming model. The blocks are arranged in a grid that is mapped to a hierarchy of CUDA cores in the GPU.

Threads can access data from multiple memory spaces. First, each thread has a private local memory and registers. Each block of threads has a shared memory visible exclusively to the threads within this block and whose lifetime equals the block's one. Finally, all threads access the same global memory space (DRAM) which is persistent across kernel launches by the same application. Because it is on-chip, shared memory is much faster than global memory, but its lifetime prevents data sharing among thread blocks. So, the shared memory is especially useful when there is reuse of data among threads in the same block, avoiding reading the same data from global memory more than once.

The NVIDIA Kepler GPU architecture [28] also provides a two-level cache hierarchy including a configurable L1 cache. There are 64 kB of on-chip memory for each multiprocessor (SMX), which can be configured as half each for the shared memory and the L1 cache, 48 kB of shared memory, and 16 kB of L1 cache or vice-versa. There is also a unified L2 cache of 1536 kB that is shared among all the SMX units.

Threads run in groups of 32, called warps, that are executed simultaneously. The occupancy is defined as the number of active warps over the maximum number of warps supported per SMX [29]. The bigger the occupancy, the larger the number of threads executing simultaneously, and therefore, the better the performance. The occupancy is limited by the number of registers used, the amount of shared memory employed, and the number of threads in a block.

To achieve the best performance in terms of execution times, several GPU programming techniques have been applied in this implementation.

1) Minimize the use of global memory and the number of data transfers between the host and device memories. In order to reduce the use of global memory, the outputs are stored over the same memory previously reserved for the inputs whenever possible (in place computation). All the computations are carried out in the GPU memory, so that the host/device memory transfers are reduced to copying the inputs and returning the outputs. Finally, data are stored in the memory so that the accesses will be coalesced whenever possible.

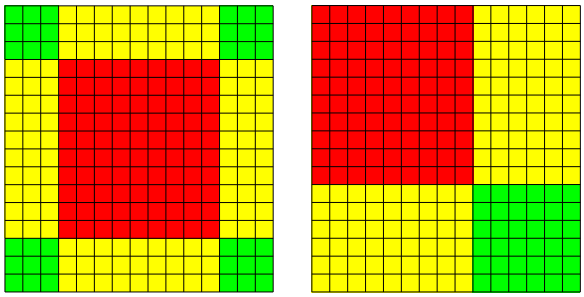

Fig. 2. Shared memory load pattern. (Left) Classical pattern. (Right) Optimized pattern.

```
Require: Hyperspectral Image, Ruleset.
 1:                                                          ▷ Preprocessing
 2:  <Create window filters>                                 ▷ GM
 3:  while iteration < maximum iterations do
 4:      <Add apron to hyperspectral image>                  ▷ GM
 5:      <Calculate distance to neighbourhood pixels>        ▷ SM
 6:      <Calculate and normalize gradients for all windows> ▷ SM
 7:                                                          ▷ Processing
 8:      <Calculate gradient distances for all windows>      ▷ GM
 9:      <Calculate gradient angles>                         ▷ GM
10:      <Select rule to apply>                              ▷ GM
11:                                                          ▷ Postprocessing
12:      <Calculate and weight output gradient>              ▷ GM
13:      <Average spectrum (filter and update hyperspectral image)> ▷ SM
14:  end while
                                  ▷ GM: Global Memory, SM: Shared Memory
```

Fig. 3. Pseudocode for the ECAS-II segmentation algorithm.

2) Optimized data load pattern in shared memory. The objective of this technique is to reduce the number of conditionals required to load data in shared memory, reducing divergence and improving the occupancy. Due to the data dependence required by the algorithm, each block needs to store in shared memory a 2-D data block bigger than the thread number. In particular, a block of threads stores the pixels corresponding to their indices in the image and also an apron. Fig. 2(left) shows an example considering a 9×10 thread-block where nine sections with different load patterns are needed to load all data (the central data, four apron borders, and four apron corners). In our implementation [see Fig. 2(right)], only four different sections with different load patterns (the main data, two larger borders, and one corner) are needed to load all data.
3) Efficient use of libraries for algebraic computations. The Magma library [30] is used to efficiently compute the matrix by matrix multiplications.
4) Search for the best kernel configuration. Performance was improved reducing the block dimensionality whenever possible (mapping 2-D data in 1-D blocks, for instance). The block size was tuned for each kernel to achieve better efficiency and minimize execution times.
5) Efficient use of the shared memory. In order to improve the efficiency of the code, the shared memory is exploited when data are reused. The configurable L1 cache/shared memory size is set to maximize the occupancy and, therefore, to reduce the execution times.

B. ECAS-II Segmenter GPU Projection

This section is devoted to explain the details of the GPU implementation of the general ECAS-II segmentation algorithm of Fig. 1. The detailed pseudocode that has been implemented can be seen in Fig. 3. Each process executed in GPU is placed between <> symbols and may involve one or more kernels. The pseudocode also includes the GM and SM acronyms to indicate kernels executed in global memory and shared memory, respectively. The process is computed entirely on GPU and a graphic representation of its computation scheme for a block of data at a given instant of time can be seen in Fig. 4.

First, a preprocessing step is computed once the inputs are stored in the GPU global memory. The first thing to do in order to execute the algorithm is to create the different window filters considered (line 2 on the pseudocode of Fig. 3). In this case, the window sizes are 3×3, 5×5, and 7×7, respectively. This means that each pixel will need access to the spectra of 49 neighboring pixels. It is worth noting than each filter contains vertical and horizontal separable components.

After this, an iterative process starts (lines 3–14) adding an apron to the hyperspectral image (line 4) in order to avoid memory access errors when reading the neighbors of the border pixels. For the window sizes considered, the apron must be a border of 3 pixels, as shown in Fig. 4.

Then, the processing stage begins. A kernel is launched in shared memory that calculates the spectral angle distances between each pixel and all of its neighbors as indicated in (1) and stores them in a matrix in global memory (line 5 in the pseudocode). In this kernel, as represented in Fig. 4, each thread processes the values for all the neighbors of a particular pixel.

The next steps (line 6) calculate the gradients for the three windows multiplying the matrix of the previously calculated distances by the corresponding filter in shared memory using the Magma library as indicated in (2), calculate the norms of these gradients, and normalize them (dividing by their norms).

Then, a process starts for each window size as indicated in Fig. 4, where three new distance matrices are computed (one for each window size) to indicate the distance of the neighborhood pixels to a line perpendicular to the gradient vectors that divides the neighborhood into two areas (line 8 in the pseudocode). After this, the angles of the gradients for each window are directly updated with respect to the angle where the average distance of the pixels within the neighborhood is smallest. The angles of the gradients are calculated (line 9) as the arctangent of the vertical and horizontal components of the gradients (3) and stored in a global memory matrix.

Once the gradient modules and angles are calculated, the process for selecting the rule to apply for each pixel can begin (line 10). In this process, the values of the pixel gradients (three modules and three angles, one for each window size) are compared to the parameters of each rule in the ruleset (4). For better performance, translations and rotations are applied in the comparison in order to make the rules invariant to these effects. Once the rule is selected, a postprocessing stage takes place where the

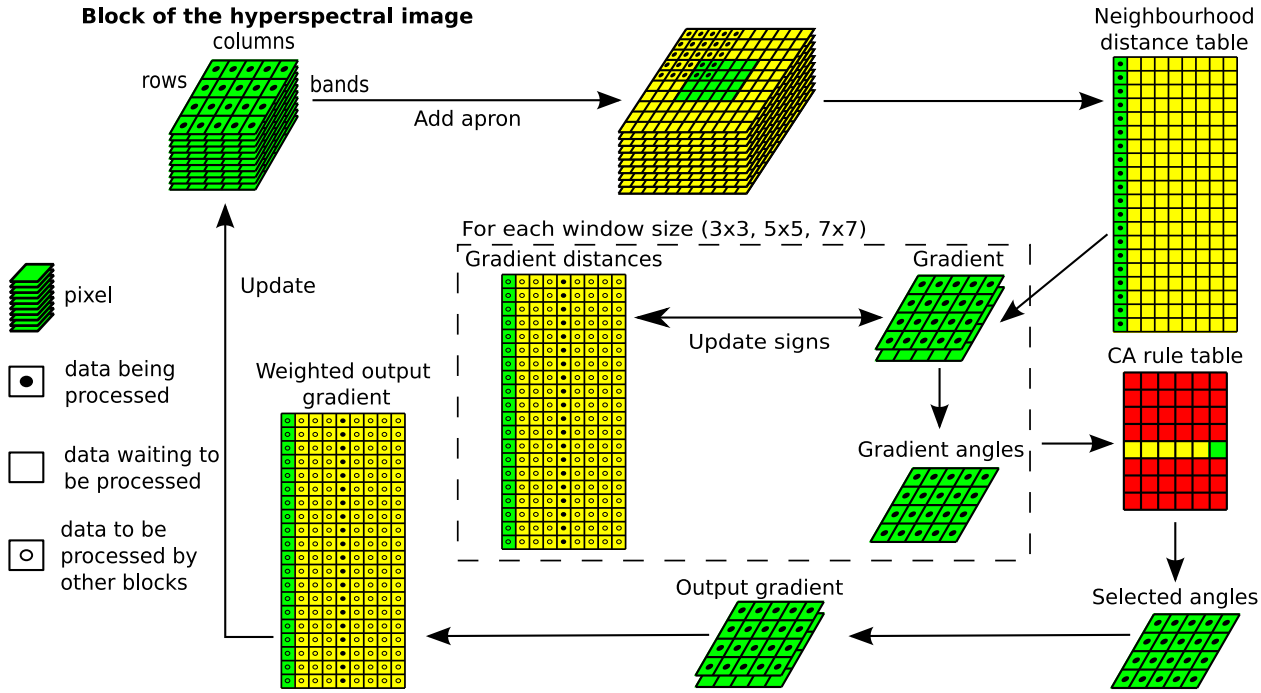

Fig. 4. GPU computation scheme of a block of threads of size 5×4, showing the data structures and the activity of each thread at a given instant of time.

pixel will be averaged with those within a radius of distance 1 moving in the direction indicated by the angle θ of the selected rule.

The output gradient to apply is calculated and stored in global memory (line 12 on pseudocode) as the cosine and sine of the angle indicated by the selected rule (for the vertical and horizontal components), thereby producing a data structure with two bands, as seen in Fig. 4. The contribution of each neighboring pixel is determined through a filter that assigns a weight inversely proportional to the distance to the central pixel. Then, the hyperspectral image is updated (line 13) as the weighted average of the spectrum of the selected neighbors for each band and for each pixel, so that no hyperspectral information is lost.

Finally, this updated image is the input to the next iteration of the algorithm. The process lasts until the selected number of iterations is reached. At the end, the last updated image is returned as the output of the algorithm.

IV. Remote Sensing Segmentation and Classification Results

This section presents some experimental results obtained by the ECAS-II segmentation algorithm along with the ELM classifier in a spectral–spatial classification scheme optimized for GPU execution in CUDA. We will start by describing the experimental conditions in Section IV-A. Then, some classification accuracy results will be detailed in Section IV-B along with the performance results in terms of execution times.

A. Hyperspectral Dataset and Experimental Conditions

The proposed algorithms have been evaluated on a PC with a quad-core Intel Core i5-3470 at 3.20 GHz and 8 GB of RAM. The code has been compiled using the gcc 4.8.4 version with OpenMP 3.0 support under Linux 14.04. Regarding the GPU implementation, the CUDA codes run on a Kepler NVIDIA GeForce GTX Titan with 14 SMXs and 192 CUDA cores each. The CUDA code has been compiled using nvcc with version 7.5 of the toolkit under Linux.

The accuracy results are expressed in terms of overall accuracy (OA), i.e., the percentage of pixels correctly classified, average accuracy (AA), i.e., the average of the percentage of pixels correctly classified per class, and kappa coefficient [31], which is the percentage of agreement corrected by the amount of agreement that could be expected due to chance alone. The performance results are expressed in terms of execution times and speedups compared to an OpenMP CPU optimized version of the algorithms parallelized using four threads and whose algebra operations are accelerated using the OpenBLAS library. We provide the average of ten independent runs for all the previously introduced metrics.

The tests were run on three hyperspectral airborne datasets [32]: A 103-band ROSIS image of the University of Pavia (Pavia Univ.) with a spatial dimension of 610×340 pixels, a 220-band AVIRIS image of 145×145 pixels taken over Northwest Indiana (Indian Pines), and a 204-band AVIRIS image of 512×217 pixels taken over the Salinas Valley, California (Salinas) [32].

The ECAS-II segmentation algorithm was applied using a ruleset comprising 20 different rules specifically evolved for the characteristics of each hyperspectral image and running a total of 15 iterations over the image to obtain the final segmentation. The distance measure employed for the filters was $1/(1 + c \cdot d)$, where c is a selected coefficient and d the distance between two pixels. The best results were obtained using values for c of 0.5

TABLE I
CLASSIFICATION ACCURACY AS PERCENTAGES

	Pavia Univ.			Indian Pines			Salinas		
	OA	AA	kappa	OA	AA	kappa	OA	AA	kappa
ECAS-II+ELM+reg	**98.43**	98.05	**97.90**	**98.67**	**98.53**	**98.42**	**97.22**	96.92	**96.28**
ECAS-II+SVM	97.51	96.12	96.71	95.77	95.47	95.18	95.85	**98.06**	95.39
ELM [21]	86.75	89.55	82.61	80.72	85.48	77.70	91.55	95.97	90.55
ELM+reg+wat con(8) [21]	95.65	95.52	94.18	92.67	94.29	91.43	93.70	96.78	92.95
V-ELM-1+reg+wat con(8) [21]	96.66	95.92	95.00	90.41	95.35	86.21	92.43	96.75	91.15
SVM+reg [33]	84.27	90.89	79.90	88.58	77.27	86.93	–	–	–
SVM+wat con(8) [11]	85.42	91.31	81.30	92.48	77.26	91.39	–	–	–
SVM+EM [33]	93.59	94.39	91.48	87.25	70.34	85.43	–	–	–
SVM+EM+reg [33]	94.68	95.21	92.02	88.83	71.90	87.24	–	–	–
HSeg+PV [12]	98.35	**98.15**	97.79	86.89	89.83	84.84	–	–	–
SAM+MV [13]	–	–	–	91.80	94.28	90.64	–	–	–
RD-MSF [14]	–	–	–	91.33	93.73	–	–	–	–

"reg" indicates that the pixelwise classifier was spatially regularized. "wat" indicates spatial processing by watershed. "con(x)" indicates connectivity of "x" neighbors.

for Pavia Univ., 0.01 for Indian Pines, and 0.05 for Salinas. In order to show the speedup of a fully exploited GPU projection, two versions of the implementation are presented. The first one (CUDA GPU GM) only uses the global memory of the GPU, whereas the second one (CUDA GPU SM) exploits the shared memory available to reduce the execution times.

The number of training samples used for the ELM classification algorithms are 200 per class, or half the number of samples in the class if there are not enough samples. These samples are randomly chosen and all the remaining samples are used for testing. The number of hidden layer neurons employed for the ELM are 500 for Pavia Univ., 950 for Indian Pines, and 350 for Salinas in all the cases [19]. A spatial regularization is applied to the ELM-based pixelwise classification as presented in [21] aiming to correct isolated misclassified pixels updating the label of the pixel if more than half of its neighbors share a different label.

B. Accuracy and Performance Results

In this section, we first address the accuracy results achieved for the proposed classification scheme and then provide some performance results. Results are shown for a spectral–spatial classification scheme such as the one described in [21] but replacing the watershed-based segmentation by the ECAS-II algorithm. We call this scheme ECAS-II+ELM+reg.

Table I shows accuracy results for the three test images in terms of OA, AA, and kappa. The best results are highlighted in bold in the table. When a result is not shown, it is because it is not available in the literature. The results are compared to a similar approach that considers SVM instead of ELM (ECAS-II+SVM). The SVM needs to optimize two parameters (C,γ) corresponding to a penalty term and the radius of the Gaussian function, respectively. In this paper, the SVMs were trained using $C = 256$ and $\gamma = 16$ for Pavia Univ., $C = 512$ and $\gamma = 0.5$ for Indian pines, $C = 512$ and $\gamma = 8$ for Salinas and the same sampling configurations used for the ELM-based one. These values were obtained by fivefold cross validation. The results are also compared to a pixelwise classifier based on ELM [21] trained

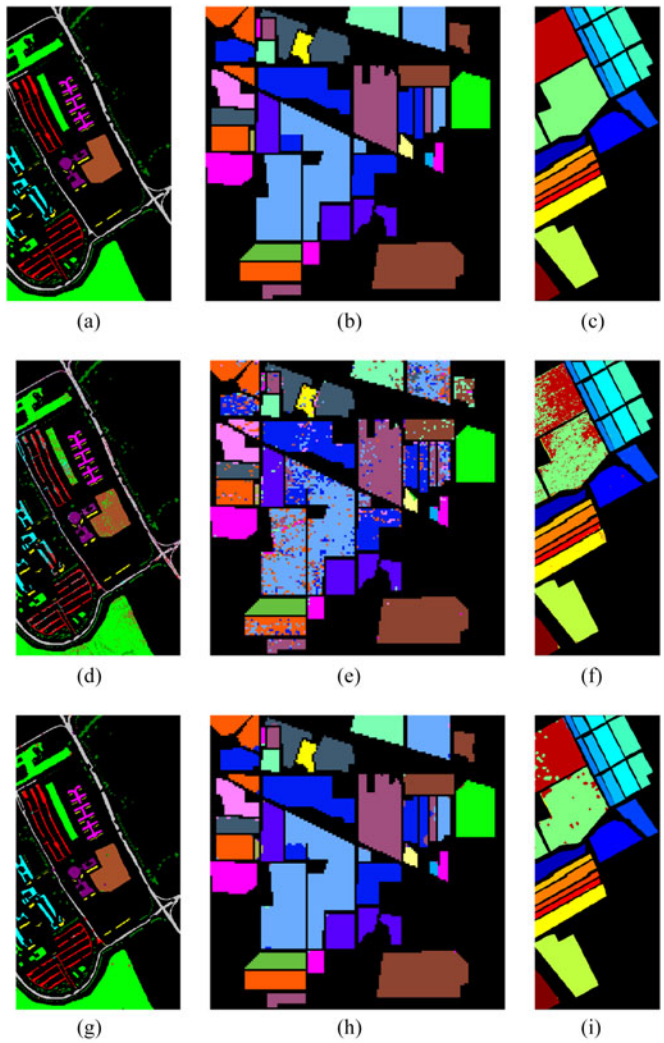

Fig. 5. Reference data for clasifications (a)–(c), ELM classification maps (d)–(f), and ECAS-II+ELM+reg classification maps (g)–(i) for the Pavia Univ. (a), (d), (g), Indian Pines (b), (d), (h), and Salinas (c), (f), (i) images.

TABLE II
SIZE (IN MEGABYTES) OF THE DATA STRUCTURES IN FIG. 4 FOR THE PAVIA UNIV. DATASET (DOUBLE DATA TYPE)

Data structure	Dimensions	Size
Image + apron (6 pixels)	$616 \times 346 \times 103$	175.624
neighborhood distances	$(610 \times 340) \times 49$	81.301
Gradients	$3 \times (610 \times 340 \times 2)$	9.955
Gradient distances	$3 \times ((610 \times 340) \times 49)$	243.902
Gradient angles	$3 \times (610 \times 340)$	4.978
CA rule table	20×6	0.001
Selected angles	610×340	1.659
Output gradient	$610 \times 340 \times 2$	3.318
Weighted output gradient	$(610 \times 340) \times 49$	81.301

under the same configurations as the ECAS-II-based schemes. Other spectral–spatial techniques available in the literature are included for comparison purposes: ELM+reg+wat con(8) [21] introduces an approach based on watershed segmentation combined with a spatially regularized ELM through a majority vote. V-ELM-1+reg+wat con(8) [21] adds the use of ensembles to the previous approach. SVM+reg [33] and SVM+wat con(8) [11] introduce similar schemes relying on SVM instead of ELM. SVM+EM [33] and SVM+EM+reg [33] use a segmentation based on partitional clustering instead of the watershed based one. Some approaches using the SAM distance to preserve all the spectral information in the segmentation process are also included: HSeg+PV [12] introduces a configuration that combines hierarchical segmentation and SVM through a plurality vote. Finally, SAM+MV [13] and RD-MSF [14] use MSF to add spatial information to an SVM through a majority vote process.

It can be seen that the implementations based on ECAS-II outperform all the different alternatives included from the literature. The configuration based on a spatially regularized ELM presented in this work achieves the better results for all the datasets. It achieves 98% of OA for the Pavia Univ. and Indian Pines datasets and 97% for the Salinas dataset and an AA also up to 98%. It can also be noticed that the approaches that preserve all the spectral information thanks to the use of the SAM distance achieve better accuracy results in general, especially in the case of the Pavia Univ. dataset for the HSeg+PV scheme. Fig. 5 shows the reference data for classifications and false color classification maps obtained by the ECAS-II+ELM+reg algorithm and also for the raw ELM for comparison purposes. It can be noticed that the spatial segmentation through the ECAS-II algorithm removes most of the isolated misclassified pixels.

Table II shows the sizes of the data structures in Fig. 4 for the case of the Pavia Univ. dataset. The maximum memory required in a given instant of time is 1.5 times the size of the dataset, corresponding to the step for calculating the gradient distances.

Regarding the performance in terms of execution times, Table III compares the execution times (in seconds) of the ECAS-II segmentation algorithm for the different images depending on the configuration chosen for the configurable shared/cache memory on the GPU, as explained in Section III-A, optimization technique 5. "Prefer SM" indicates the case when 48 kB

TABLE III
ECAS-II SEGMENTER EXECUTION TIMES (IN SECONDS) FOR PAVIA UNIV., INDIAN PINES, AND SALINAS IMAGES FOR THE DIFFERENT L1-SM CONFIGURATIONS

	Pavia Univ.	Indian Pines	Salinas
Prefer SM	**13.31**	**3.21**	**12.89**
Prefer L1	30.05	7.27	30.62
Prefer none	13.32	3.21	12.89

are dedicated to shared memory and 16 kB for L1 cache and "Prefer L1" indicates the opposite. "Prefer none" is for the case when 32 kB are devoted to each one. As stated before, the application needs a large amount of shared memory, that is why, the execution times are notably smaller for every image when we assign more shared memory than L1 cache. Regarding this, the remaining results will follow the "Prefer SM" configuration.

Table IV details the execution times and speedups of each step of the algorithm for the three datasets both in CPU (OpenMP) and GPU (CUDA) where shared memory is used. It also shows the percentage of the total CPU time (in average for the three datasets) for each step. "TI" indicates total time per iteration, i.e., the sum of the time for the different steps included in Table IV. The steps to calculate the neighborhood distances, select rule, and average spectrum (lines 5, 10, and 13 in Table IV and in the pseudocode of Fig. 3, respectively) cover 93% of the execution time of each iteration. These three steps are, therefore, the most relevant ones to be optimized.

In particular, the kernel to calculate the distance to the neighborhood pixels (line 5) obtains a high speedup because of the data load pattern, the kernel configuration, and the efficient use of shared memory (Section III-A, optimization techniques 2, 4, and 5). As its operation involves data reuse when accessing the neighborhood of each pixel, the shared memory of the GPU is fully exploited to achieve better speedups. The kernel works with 2-D blocks of data in the spatial dimension. For this reason, the bigger the spatial dimensions of the image, the better the speedup. Achieving the best result of $7.2\times$ for the Pavia Univ. dataset. The theoretical occupancy is 43%, limited by the amount of shared memory needed and the achieved occupancy is between 41% and 42% for the three datasets.

For the average spectrum kernel (line 13 in Table IV and in pseudocode of Fig. 3), GPU optimization techniques 2, 4, and 5 (see Section III-A) are applied. The kernel involves the reuse of neighborhood pixels, and therefore, the use of shared memory allows improving the efficiency of the kernel. It works with 3-D blocks of data, which allows a better speedup the bigger the dimensions of the image, both spectral and spatial. As shown in Table IV, the Salinas dataset achieves the best speedup for this kernel ($90.6\times$). The occupancy achieved in this kernel is between 72% and 73%, being the maximum theoretical occupancy 75%, limited by the amount of shared memory available.

Finally, the kernel employed for selecting the optimum rule for each pixel (line 10) exploits the faster accesses to the registers, and its speedup is based on the optimization techniques 1 and 4 described in Section III-A. This kernel computes a

TABLE IV
ECAS-II SEGMENTER DETAILED EXECUTION TIMES (IN SECONDS) AND SPEEDUPS OPENMP CPU–CUDA GPU BY STEP (FOR ONE ITERATION OF THE ALGORITHM) FOR PAVIA UNIV., INDIAN PINES, AND SALINAS IMAGES

		Pavia Univ.			Indian Pines			Salinas		
Line	% of CPU time	CPU	GPU	Speedup	CPU	GPU	Speedup	CPU	GPU	Speedup
4	0.67%	0.0965	0.0026	37.8×	0.0158	0.0009	17.2×	0.0931	0.0029	31.8×
5	28.48%	3.5691	0.4978	7.2×	0.7724	0.1488	5.2×	3.7583	0.5490	6.8×
6	2.55%	0.5053	0.0270	18.7×	0.0502	0.0030	16.5×	0.2643	0.0150	17.6×
8	2.34%	0.4288	0.0786	5.5×	0.0513	0.0122	4.2×	0.2403	0.0440	5.5×
9	0.03%	0.0068	0.0008	8.5×	0.0007	0.0001	6.3×	0.0035	0.0007	5.3×
10	12.09%	2.3965	0.1196	20.0×	0.2380	0.0124	19.2×	1.2578	0.0658	19.1×
12	0.35%	0.0720	0.0112	6.4×	0.0065	0.0019	3.5×	0.0376	0.0063	5.9×
13	52.70%	6.5618	0.1273	51.5×	0.8227	0.0303	27.1×	12.6365	0.1395	90.6×
TI		13.6368	0.8650	15.8×	1.9576	0.2096	9.3×	18.2913	0.8233	22.2×

"line" refers to the corresponding line in Fig. 3. "TI" indicates the total time for one iteration. "% of CPU time" represents the percentage of "TI" in CPU for each step of the algorithm.

TABLE V
ECAS-II SEGMENTER EXECUTION TIMES (IN SECONDS) FOR PAVIA UNIV., INDIAN PINES, AND SALINAS IMAGES

	Pavia Univ.	Indian Pines	Salinas
OpenMP CPU	206.57	29.69	275.16
CUDA GPU GM	16.45	3.34	15.72
CUDA GPU SM	13.31	3.21	12.89
Speedup CPU-GPU GM	12.6×	8.9×	17.5×
Speedup CPU-GPU SM	**15.5**×	**9.2**×	**21.4**×

fixed-size set of rules. It works in 1-D blocks of threads comparing simultaneously each pixel gradients against the ruleset that is stored in local memory and registers to minimize access times. It achieves a speedup between 19× and 20×, and the execution time is higher the bigger the spatial dimensions of the image. In this case, the theoretical occupancy is limited by the number of registers needed to 25%, achieving, in practice, an occupancy of 24.9%.

The global performance results in terms of execution times (in seconds) and speedups calculated over the OpenMP multicore implementations for four threads are shown in Table V. Two GPU implementations are presented in the table, including the time needed to transfer data to the global memory of the GPU. The first one in the table, CUDA GPU GM stands for an implementation fully executed in global memory. On the other hand, CUDA GPU SM stands for the implementations indicated in Fig. 4, where the faster shared memory is exploited as much as possible. The GPU implementation achieves large speedups for the three datasets. Execution times show that the GPU versions achieve better speedup the larger the dimensions of the image. They also show that the shared memory version (CUDA GPU SM in Table V), as expected, achieves better speedup than the global memory version (CUDA GPU GM in the table) the larger the spatial dimensions of the image, achieving a maximum speedup of 21.4 × for the Salinas image. The reason is a better exploitation of the high number of computing threads available that simultaneously process each spatial pixel of the image. If the entire spectral–spatial classification scheme is considered, the ELM algorithm is much faster than the SVM with speedups of up to 8.8× with respect to the SVM as presented in [21].

V. CONCLUSION

In this work, we present an efficient GPU projection of the ECAS-II segmenter that allows the segmentation of hyperspectral images without loss of spectral information. The algorithm is combined with an ELM in order to obtain a final spectral–spatial classification map. The accuracies achieved using this approach show better results than other state-of-the-art techniques, reaching 98% of OA for the Pavia Univ. and Indian Pines datasets and 97% for the Salinas dataset. Furthermore, the execution times for the segmentation algorithm are notably better in the GPU projection, achieving speedups of up to 21.4× compared to an OpenMP CPU version of the algorithm.

REFERENCES

[1] F. D. Van der Meer *et al.*, "Multi- and hyperspectral geologic remote sensing: A review," *Int. J. Appl. Earth Observ. Geoinf.*, vol. 14, no. 1, pp. 112–128, 2012.

[2] J. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot, "Hyperspectral remote sensing data analysis and future challenges," *IEEE Geosci. Remote Sens. Mag.*, vol. 1, no. 2, pp. 6–36, Jun. 2013.

[3] M. Fauvel, Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Advances in spectral-spatial classification of hyperspectral images," *Proc. IEEE*, vol. 101, no. 3, pp. 652–675, Mar. 2013.

[4] Y. Tarabalka, J. A. Benediktsson, J. Chanussot, and J. C. Tilton, "Multiple spectral–spatial classification approach for hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 48, no. 11, pp. 4122–4132, Nov. 2010.

[5] B. Priego, F. Bellas, and R. J. Duro, "Evolving cellular automata to segment hyperspectral images using low dimensional images for training," *Bioinspired Comput. Artif. Syst.*, vol. 9108, pp. 117–126, 2015.

[6] J. A. Benediktsson, J. A. Palmason, and J. R. Sveinsson, "Classification of hyperspectral data from urban areas based on extended morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 3, pp. 480–491, Mar. 2005.

[7] P. Ghamisi, J. A. Benediktsson, and M. O. Ulfarsson, "Spectral–spatial classification of hyperspectral images based on hidden Markov random fields," *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 5, pp. 2565–2574, May 2014.

[8] M. Dalla Mura, A. Villa, J. A. Benediktsson, J. Chanussot, and L. Bruzzone, "Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 8, no. 3, pp. 542–546, May 2011.

[9] L. M. Bruce, C. H. Koger, and J. Li, "Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 10, pp. 2331–2338, Oct. 2002.

[10] P. Quesada-Barriuso, F. Argüello, D. B. Heras, and J. A. Benediktsson, "Wavelet based classification of hyperspectral images using extended morphological profiles on graphics processing units," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2962–2970, Jun. 2015.

[11] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson, "Segmentation and classification of hyperspectral images using watershed transformation," *Pattern Recog.*, vol. 43, no. 7, pp. 2367–2379, 2010.

[12] J. C. Tilton, Y. Tarabalka, P. M. Montesano, and E. Gofman, "Best merge region-growing segmentation with integrated nonadjacent region object aggregation," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 11, pp. 4454–4467, Nov. 2012.

[13] Y. Tarabalka, J. Chanussot, and J. A. Benediktsson, "Segmentation and classification of hyperspectral images using minimum spanning forest grown from automatically selected markers," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 5, pp. 1267–1279, Oct. 2010.

[14] K. Bernard, Y. Tarabalka, J. Angulo, J. Chanussot, and J. A. Benediktsson, "Spectral–spatial classification of hyperspectral data based on a stochastic minimum spanning forest approach," *IEEE Trans. Image Process.*, vol. 21, no. 4, pp. 2008–2021, Apr. 2012.

[15] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 8, pp. 1778–1790, Aug. 2004.

[16] M. Fauvel, J. A. Benediktsson, J. Chanussot, and J. R. Sveinsson, "Spectral and spatial classification of hyperspectral data using SVMs and morphological profiles," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 11, pp. 3804–3814, Nov. 2008.

[17] Y. Tarabalka, J. A. Benediktsson, J. Chanussot, J. Angulo, and M. Fauvel, "Classification of hyperspectral data using support vector machines and adaptive neighborhoods," in *Proc. 6th EARSeL SIG IS Workshop*, pp. 1–6, 2009.

[18] M. Pal, "Extreme-learning-machine-based land cover classification," *Int. J. Remote Sens.*, vol. 30, no. 14, pp. 3835–3841, 2009.

[19] D. B. Heras, F. Argüello, and P. Quesada-Barriuso, "Exploring ELM-based spatial–spectral classification of hyperspectral images," *Int. J. Remote Sens.*, vol. 35, no. 2, pp. 401–423, 2014.

[20] F. Argüello and D. B. Heras, "Elm-based spectral–spatial classification of hyperspectral images using extended morphological profiles and composite feature mappings," *Int. J. Remote Sens.*, vol. 36, no. 2, pp. 645–664, 2015.

[21] J. López-Fandiño, P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 8, no. 6, pp. 2884–2893, Jun. 2015.

[22] C. Lee *et al.*, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[23] P. Quesada-Barriuso, F. Argüello, and D. B. Heras, "Computing efficiently spectral-spatial classification of hyperspectral images on commodity GPUs," in *Recent Advances in Knowledge-based Paradigms and Applications*, vol. 234. Switzerland: Springer, 2014, pp. 19–42.

[24] P. Quesada-Barriuso, D. B. Heras, and F. Argüello, "Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata," in *Proc. IEEE 10th Int. Symp. Parallel Distrib. Process. Appl.*, 2012, pp. 79–86.

[25] N. Keshava, "Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 7, pp. 1552–1565, Jul. 2004.

[26] P. E. Dennison, K. Q. Halligan, and D. A. Roberts, "A comparison of error metrics and constraints for multiple endmember spectral mixture analysis and spectral angle mapper," *Remote Sens. Environ.*, vol. 93, no. 3, pp. 359–367, 2004.

[27] C. Chen, J. Jiang, B. Zhang, W. Yang, and J. Guo, "Hyperspectral image classification using gradient local auto-correlations," in *Proc. 3rd IAPR Asian Conf. Pattern Recog.*, 2015, pp. 454–458.

[28] *NVIDIA Kepler GK110 Architecture Whitepaper*, Nvidia, Santa Clara, CA, USA, 2012.

[29] M. Harris, "Optimizing cuda," presented at the SC07 Int. Conf. for High Performance Computing, Networking, Storage and Analysis, Reno-Sparks. United States of America Nov. 10-16, 2007.

[30] S. Tomov, R. Nath, P. Du, and J. Dongarra, *MAGMA Users' Guide*, Innovative Comput. Lab., Univ. Tennessee, Knoxville, TN, USA, 2011.

[31] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis*, vol. 3. New York, NY, USA: Springer, 1999.

[32] C. I. G. from the Basque University (UPV/EHU), "Hyperspectral remote sensing scenes," 2014. [Online]. Available: http://www.ehu.es

[33] Y. Tarabalka, J. A. Benediktsson, and J. Chanussot, "Spectral–spatial classification of hyperspectral imagery based on partitional clustering techniques," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2973–2987, Aug. 2009.